

REST in Practice

RESTful Web Services: Principles, Patterns, Emerging Technologies

[./]

Tutorial at ICWE 2010

[<http://icwe2010.webengineering.org/>] (Vienna,
Austria)

[Erik Wilde](#) ([UC Berkeley School of Information](#))

July 6, 2010



[<http://creativecommons.org/licenses/by/3.0/>]

[This work is licensed under a CC
Attribution 3.0 Unported License](#) [<http://creativecommons.org/licenses/by/3.0/>]

[Erik Wilde: REST in Practice](#)

[Contents](#)

Contents

| | |
|--|----|
| • Abstract | 2 |
| • Precious Snowflakes | 3 |
| • RESTful "Hello World" | 4 |
| • Feeds! | 5 |
| • Syndication | 6 |
| • 1 Syndication Formats | |
| ◦ 1.1 RSS | |
| ▪ RSS History | 9 |
| ▪ RSS 2.0 | 10 |
| ▪ RSS 2.0 Example | 11 |
| ▪ Consuming RSS | 12 |
| ▪ RSS Political Problems | 13 |
| ◦ 1.2 Atom | |
| ▪ Atom History | 15 |
| ▪ Atom vs. RSS | 16 |
| ▪ Atom Example | 17 |
| ▪ Atom Content | 18 |
| ▪ Atom Content Examples | 19 |
| ▪ Atom Categories | 20 |
| ▪ Switching from RSS to Atom | 21 |
| • 2 Syndication Aggregation | |
| ◦ End-User Aggregation | 23 |
| ◦ Aggregation Intermediaries | 24 |

| | |
|--|----|
| ○ 2.1 FeedBurner | |
| ▪ Fixing Feeds | 26 |
| ▪ Load Balancing | 27 |
| ▪ Statistics/Analytics | 28 |
| ▪ Query Capabilities | 29 |
| ▪ Supporting Queryable Feeds | 30 |
| • 3 Atom Publishing Protocol (AtomPub) | |
| ○ RESTful Syndication | 32 |
| ○ Collections, Members, Entries, Media | 33 |
| ○ Atom/AtomPub Data Model | 34 |
| ○ Protocol Summary | 35 |
| ○ Collection Management | 36 |
| ○ Service Documents | 37 |
| ○ Service Document Example | 38 |
| ○ Category Documents | 39 |
| ○ Category Document Example | 40 |
| • 4 Extending Atom/AtomPub | |
| ○ Atom/AtomPub as Foundation | 42 |
| ○ RESTful Web Service Design | 43 |
| ○ RESTful Web Service Design Procedure | 44 |
| ○ Watch Caching | 45 |
| • 5 REST Tools | |
| ○ Toolboxes and Frameworks | 47 |
| ○ REST Frameworks | 48 |
| ○ Poster | 49 |

[Erik Wilde: REST in Practice](#)

Abstract

(2)

REST's level of abstraction and its simplicity as a small set of constraints can make it hard to get a grasp on how it can be applied for real-world projects. This presentation introduces *real-world REST* by looking at how REST can be used by reusing existing RESTful designs in terms of representations and interaction protocols; *Atom* and the *Atom Publishing Protocol (AtomPub)* are used as examples for existing RESTful designs. In addition, we take a brief look at how to go beyond using these "canned REST" approaches, and how existing programming frameworks provide support for designing and implementing RESTful services.

Precious Snowflakes

(3)

- Many applications are not as unique as you might think
 - reusing design patterns and even technologies is a good idea
 - avoiding reuse usually should be justified by good reasons
- Take the Web's HTML as an inspiring example
 - HTML is not all that great as a document format
 - it was good enough as a container for a lot of useful content
 - the *network effect* far outweighs its *functional shortcomings*
 - could you imagine a Web based on PDF or Word?
- Simplicity and wide applicability are valuable
 - *simplicity* means a lower barrier to entry
 - *wide applicability* means higher chances to create network effects



RESTful “Hello World”

(4)

- Internet connectivity establishes HTTP URI addressability
 - host must be reachable and identifiable (DNS name or at least IP address)
- HTTP server provides the uniform interface (GET only)
 - by default listens to incoming HTTP requests on port 80
 - in simple scenarios often is configured to (also) serve static files
- XML and `application/xml` provide a machine-readable representation

```
<xml>Hello World</xml>
```

- Dropping the XML into the file system creates a RESTful “Hello World” service
 - [hello-world.xml](#) [./src/hello-world.xml]
- Accessible via any HTTP- and XML-capable client

[Erik Wilde: REST in Practice](#)

Feeds!

(5)

- Information dissemination is a complex and varied problem
- Feeds look at all these problems with a simple approach
 - the simple approach lacks some sophistication/specialization
 - less specialization means wider applicability
 - wider applicability means bigger network effects
 - pull-based syndication is loosely coupled and scalable
- Some claim that pull publishing is bad and *Publish/Subscribe (PubSub)* is good
 - there is little proof that this is more than just a claim
 - scalable PubSub is expensive to implement (but still may make sense)
 - PubSub may be a good idea if the requirements rule out REST



7 of 45

7/4/2010 19:51

[Erik Wilde: REST in Practice](#)

Syndication

(6)

- Atom is a format and also [an evolving landscape](http://dret.typepad.com/dretblog/atom-landscape.html) [http://dret.typepad.com/dretblog/atom-landscape.html]
 - [Atom](#) [Atom (1)], the feed format
 - [Atom Publishing Protocol \(AtomPub\)](#) [Atom Publishing Protocol (AtomPub) (1)], write access to Atom-oriented collections
 - support for threaded discussions in feeds
 - feed licensing
 - feed paging and archiving
 - [many other ongoing developments](http://dret.typepad.com/dretblog/atom-landscape.html) [http://dret.typepad.com/dretblog/atom-landscape.html]
- A landscape of RESTful interactions with "collections of things"

8 of 45

7/4/2010 19:51

Syndication Formats

RSS

RSS History

(9)

- “[The Myth of RSS Compatibility](http://diveintomark.org/archives/2004/02/04/incompatible-rss) [http://diveintomark.org/archives/2004/02/04/incompatible-rss]” provides a good overview
- RSS is a schoolbook example for “why standards are a good thing”
 - [[@rss09](#)] was created for the *My Netscape* portal in March 1999
 - RSS 0.91 (a simplification) was introduced in July 1999 (as an interim solution)
 - the AOL/Netscape merger removed the format from the company’s portal
 - RSS was without an owner, and different parties claimed/denied ownership
 - [[@rss10](#)] was created by an informal developer group
 - RSS 0.92 (and 0.93 and 0.94) were published without acknowledging RSS 1.0
 - finally, [RSS 2.0](#) [RSS 2.0 (1)] was released as a follow-up to the RSS 0.9x versions
- Using RSS has become an exercise in managing a menagerie of versions

RSS 2.0

(10)

- RSS now means *Really Simple Syndication*
 - RSS 2.0 is the continuation of the 0.91 branch (which dropped RDF)
 - together with [[@rss10](#)] it is the most popular version of RSS
 - migration from 0.91 to 2.0 is easily possible
- RSS 2.0 tries to avoid the use of XML Namespaces
- RSS 2.0 is [increasingly used with extensions](http://rss-extensions.org/wiki/Main_Page) [http://rss-extensions.org/wiki/Main_Page] for vendor-specific information
 - the RSS core is minimal, so many applications need extensions
 - many extensions have overlapping functionality
 - most extensions have unclear semantics and unclear versioning policies

RSS 2.0 Example

(11)

```
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel>
    <title>XML.com</title>
    <link>http://www.xml.com/</link>
    <description>XML.com features a rich mix of information and services for the XML community.
  </description>
  <language>en-us</language>
  <item>
    <title>Normalizing XML, Part 2</title>
    <link>http://www.xml.com/pub/a/2002/12/04/normalizing.html</link>
    <description>In this second and final look at applying relational normalization techniques
to W3C XML Schema data modeling, Will Provost discusses when not to normalize, the scope of
uniqueness and the fourth and fifth normal forms.</description>
    <dc:creator>Will Provost</dc:creator>
    <dc:date>2002-12-04</dc:date>
  </item>
```

Consuming RSS

(12)

- RSS feeds often have quality problems
 - surprisingly often feeds do not even deliver well-formed XML
 - the use of embedded markup in RSS is not well-defined ("parse and pray")
- Writing an RSS reader from scratch is not a good idea
- There are three major tasks which RSS readers must do
 1. accept non-XML RSS feeds and fix them to be XML
 2. look at the feed contents and bring them into a unified form
 3. produce a unified view of feeds regardless of the RSS version

RSS Political Problems

(13)

- Multiple and incompatible [RSS History](#) [RSS History (1)] are still in widespread use
 - [[@rss10](#)] and [RSS 2.0](#) [RSS 2.0 (1)] are “incompatible by design” (RDF vs. non-RDF)
 - none of the RSS versions is maintained by a universally accepted standards body
- None of the specifications is being updated or fixed
 - some of the lessons learned by RSS deployment are not used in a new version
 - it is unlikely that a new version will be produced which merges the RSS landscape
- Invent something new instead of trying to fix RSS
 - [Atom](#) [Atom (1)] started in 2003 (called *Echo* at first)
 - W3C or IETF would have been promising candidates for a “new RSS”
 - W3C is more formal, IETF is more developer-centered
 - [IETF was chosen over W3C](#) [http://www.bestkungfu.com/?p=492] because of the Atom community's preferences

Atom

Atom History

(15)

- RSS's shortcomings were very apparent and could not be fixed
- In mid-2003, discussions started about an improved format
- It also became apparent that the format should have a protocol
- Atom 0.3 was released in December 2003 but had no formal home
- IETF was chosen as the new home with a working group in June 2004
- [RFC 4287](#) [http://dret.net/rfc-index/reference/RFC4287] was published in December 2005
- [AtomPub](#) [Atom Publishing Protocol (AtomPub) (1)] has been published as [RFC 5032](#) [http://dret.net/rfc-index/reference/RFC5032] in October 2007



Atom vs. RSS

(16)

- Standardized by the IETF (well-defined process)
- Classification of entries (user-defined categories)
- More XML-like markup design (more nesting)
- Namespaces are used and supported as standard mechanism
- Atom feeds *must* be well-formed XML (there even [is a schema](http://atompub.org/2005/08/17/atom.rnc))
- Interpretation of content is well-defined (various content types)
- Support for xml:lang and xml:base

Atom Example

(17)

```
<feed xmlns="http://www.w3.org/2005/Atom" xml:lang="en-us">
  <title>ongoing</title>
  <id>http://www.tbray.org/ongoing/</id>
  <link rel='self' href="http://www.tbray.org/ongoing/ongoing.atom"/>
  <updated>2007-04-11T12:55:09-07:00</updated>
  <author>
    <name>Tim Bray</name>
  </author>
  <subtitle>ongoing fragmented essay by Tim Bray</subtitle>
  <entry xml:base="When/200x/2007/04/02/">
    <title>Atom Publishing Protocol Interop!</title>
    <id>http://www.tbray.org/ongoing/When/200x/2007/04/02/APP-Interop</id>
    <published>2007-04-02T13:00:00-07:00</published>
    <updated>2007-04-10T14:24:00-07:00</updated>
    <category scheme="http://www.tbray.org/ongoing/What/" term="Technology/Atom"/>
    <category scheme="http://www.tbray.org/ongoing/What/" term="Technology"/>
    <category scheme="http://www.tbray.org/ongoing/What/" term="Atom"/>
    <content type="xhtml">
      <div xmlns="http://www.w3.org/1999/xhtml">
        <p>Mark your calendar: <a href="http://www.intertwingly.net/wiki/pie/
        /April2007Interop">April 16-17 at Google</a>. <em>Everybody</em> is invited, provided they
        bring along an APP implementation, client or server. This was just announced a couple of days
        ago, and as I write this there are already <s>six</s> twelve client and <s>seven</s> fourteen
        server implementations signed up to be there and try to <a href="http://www.intertwingly.net
        /wiki/pie/InteropGrid">fill in the grid</a>. Let's drop some names, in alphabetical order: AOL,
        Flock, Google, IBM, Lotus, Microsoft, Oracle, O'Reilly, Six Apart, Sun, WordPress. Um, have I
        mentioned that the APP is going to be huge?</p>
      </div>
    </content>
  </entry>
</feed>
```

```
</entry>  
</feed>
```

Atom Content

(18)

- RSS had no safe way of finding out what an entry's content is
 - this led to different implementations being "smart" about what the RSS author really wanted
 - one of Atom's main goals was to improve this in a well-defined way
 - Atom allows escaped markup (the only way to include non-XML HTML in an XML format)
- Each content element should have a type (the default is text)
- Atom's content interpretation algorithm (use first applicable rule):
 1. if type is text, no child elements are allowed (plain text content)
 2. if type is html then RSS's method of escaped markup is used
 3. if type is xhtml then there must be an div containing XHTML markup
 4. if type is an XML media type then the content should be treated as this type
 5. if type starts with text/ then no child elements are allowed
 6. for all other values, the content must be an base64-encoded entity of the specified MIME type

Atom Content Examples

(19)

```
<content type="xhtml">
  <div xmlns="http://www.w3.org/1999/xhtml">
    One <strong>bold</strong> foot forward
  </div>
</content>
```

[http://www.xml.com/lpt/a/1633]

```
<content>The "atom:content" element either contains or links to the content of the entry. The
content of atom:content is Language-Sensitive.</content>
```

[http://www.xml.com/lpt/a/1633]

```
<content type="html">The &lt;code>atom:content&lt;/code> element either contains or links to
the content of the entry. The content of &lt;code>atom:content&lt;/code> is &lt;a
href="http://www.ietf.org/rfc/rfc3066.txt">Language-Sensitive&lt;/a>.</content>
```

[http://www.xml.com/lpt/a/1633]

```
<content type="image/png">
iVBORw0KGgoA ... TAAAAAE1FTkSuQmCC
</content>
```

[http://www.xml.com/lpt/a/1633]

```
<content src="image.png" type="image/png"/>
```

[http://www.xml.com/lpt/a/1633]

Atom Categories

(20)

- Atom allows to assign categories to entries
 - each category element must have a term attribute for the category
 - an optional scheme identifies the categorization scheme (ontology, taxonomy, ...)
 - an optional label attribute provides a human-readable label for the category
- [AtomPub](#) [Atom Publishing Protocol (AtomPub) (1)] defines a document format for [Category Documents](#) [Category Documents (1)]
- Three different cases of categorization can be distinguished
 1. use a well-known scheme (such as *Dublin Core*)
 2. use a private but well-designed scheme (which has a URI and can be reused reliably)
 3. use tags without schemes, which then are little more than content labels

Switching from RSS to Atom

(21)

- Generate both feeds but serve RSS with an HTTP redirect (301)
 - old subscribers with broken clients can still use the RSS feed
 - old subscribers with correct clients will use the Atom feed
- Atom exposes more information than RSS (category for tags)
 - the mapping of publishing info to the feed has to be changed/extended
 - for standard metadata use Atom's built-in metadata elements
 - for application-specific metadata consider reusing an existing metadata schema
- Atom can be used to publish snippets as well as full content
 - content allows any type of content to be used and may contain a complete entry
 - summary allows only text and should provide a condensed version of an entry
 - some Atom sources publish two feeds for summaries and content
- Generate good Atom and downgrade it to RSS 1.0 & 2.0

Syndication Aggregation

End-User Aggregation

(23)

- Users often have a small number of preferred Web sites
 - news can be tracked by subscribing to their feeds
 - this requires a feed-aware client (Firefox is not a good RSS reader)
- How can users find a feed?
 - feeds have URIs (they are Web resources) and can be referenced from within HTML



```
<link rel="alternate" type="application/rdf+xml" title="..." href="..." />
<link rel="alternate" type="application/rss+xml" title="..." href="..." />
<link rel="alternate" type="application/atom+xml" title="..." href="..." />
```

Aggregation Intermediaries

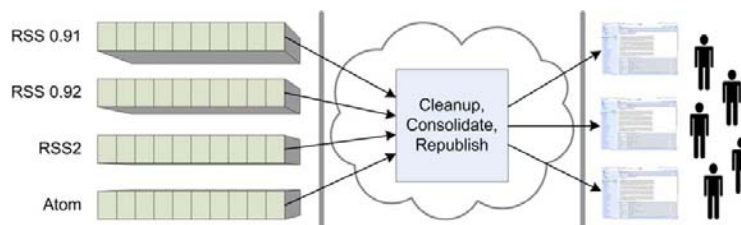
(24)

- RSS is a frequently used format between content providers
 - news agencies want to distribute news items as easy as possible
 - by using a single format, producers and consumers can more easily interoperate
- The [RSS History](#) [RSS History (1)] caused publishers to look for something better
 - [RSS](#) [RSS (1)] had a head start and still is widely used
 - [Atom](#) [Atom (1)] is a much better format and will eventually replace RSS
- User-configured portals are the merger between both scenarios
 1. users select a number of feeds as their preferred information sources
 2. the feeds are presented on a [single personalized Web page](http://www.google.com/reader/view/) [http://www.google.com/reader/view/]
 3. by [sharing](http://www.google.com/reader/shared/16601496766743088901) [http://www.google.com/reader/shared/16601496766743088901] and [re-publishing](http://www.google.com/reader/public/atom/user/16601496766743088901/state/com.google/broadcast) [http://www.google.com/reader/public/atom/user/16601496766743088901/state/com.google/broadcast] items, users are becoming aggregation intermediaries

FeedBurner

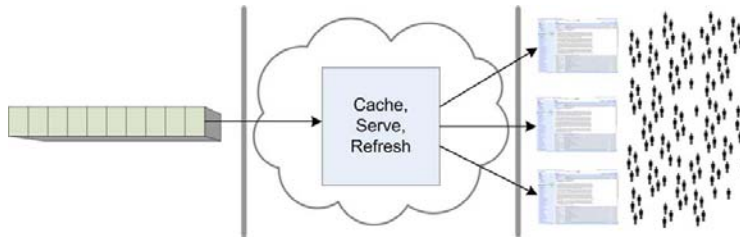
Fixing Feeds

(26)



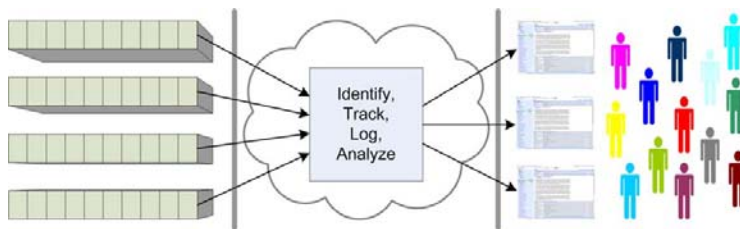
Load Balancing

(27)



Statistics / Analytics

(28)



Query Capabilities

(29)

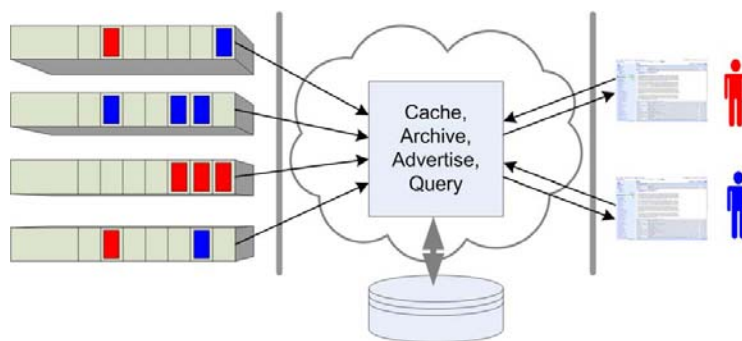
- Feed technology is still evolving
- Feeds are mostly viewed as ordered by time
 - allows optimization for accesses and caching
 - makes it hard to use feeds for non-timed information
- Feeds could be ordered by any sort key
 - makes server-side feed processing much more expensive
 - enables customized feeds that are processed on the server-side

27 of 45

7/4/2010 19:51

Supporting Queryable Feeds

(30)



28 of 45

7/4/2010 19:51

Atom Publishing Protocol (AtomPub)

RESTful Syndication

(32)

- Atom is a format for retrieving a set of entries as a feed document
 - feeds often are time-based and are refreshed periodically or whenever needed
 - feeds can use any other strategy for deciding what to publish
- Read-only access to feeds should be complemented by full access
 - full access needs the "CUD" out of the "CRUD" set of operations
 - many Web-centric technologies try to build on the Web's REST model of interaction
- AtomPub builds on Atom and adds a RESTful protocol on top of it
 - POST for creating new entries (sending the request to the collection)
 - PUT for updating existing entries (overwriting the existing entry)
 - DELETE for deleting entries from a collection

29 of 45

7/4/2010 19:51

Collections, Members, Entries, Media

(33)

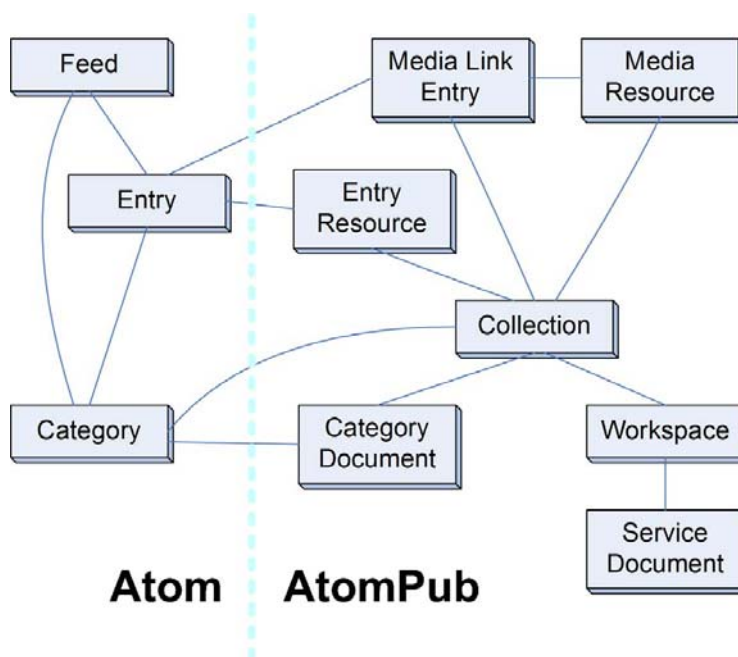
- AtomPub's top-level concept is a *collection*
 - collections are used for managing and organizing *members*
 - Atom feed documents are the representation of collections
 - collections just exist; there is no way of interacting with them
- Members of a collection can be *entry* and *media* resources
 - entry resources represent metadata and are represented as Atom entries
 - media resources can have any media type and are the data described by entries
 - a *media link entry* is an entry associated with a member

30 of 45

7/4/2010 19:51

Atom/AtomPub Data Model

(34)



Protocol Summary

(35)

| Resource | HTTP Method | Representation | Description |
|---------------|-------------|---|--|
| Introspection | GET | Atom Service Document [Service Documents (1)] | Enumerates a set of collections and lists their URIs and other information about the collections |
| Collection | GET | Atom Feed | A list of member of the collection (this may be a subset of all entries in the collection) |
| Collection | POST | Atom Entry | Create a new entry in the collection |
| Member | GET | Atom Entry | Get the Atom Entry |
| Member | PUT | Atom Entry | Update the Atom Entry |
| Member | DELETE | n/a | Delete the Atom Entry from the collection |

Collection Management

(36)

- Managing collections is outside the scope of AtomPub
 - AtomPub allows interactions with existing collections
 - Collection management is inaccessible or using a proprietary protocol
- “Metacollections” (collections of collections) can extend AtomPub
 - each member in a metacollection corresponds with a collection
 - accessing a metacollection means managing collections
 - AtomPub's existing features can be reused for collection management
- Ongoing discussion in the Atom community
 - currently there is no widely established method

Service Documents

(37)

Service Documents represent server-defined groups of Collections, and are used to initialize the process of creating and editing resources.

- The “real” top-level construct of AtomPub is the *workspace*
 - collections on a server are organized into different workspaces
 - workspaces have no AtomPub semantics and no operations can be performed on them
- Service documents list constraints on the members of collections
 - `accept` specifies a comma-separated list of media ranges (with entry as special value)
 - `categories` defines the list of categories that can be applied to members (can be fixed)
 - AtomPub servers are likely to reject operations not satisfying these constraints

Service Document Example

(38)

```
<service xmlns="http://purl.org/atom/app#" xmlns:atom="http://www.w3.org/2005/Atom">
  <workspace>
    <atom:title>Main Site</atom:title>
    <collection href="http://example.org/reilly/main">
      <atom:title>My Blog Entries</atom:title>
      <categories href="http://example.com/cats/forMain.cats"/>
    </collection>
    <collection href="http://example.org/reilly/pic">
      <atom:title>Pictures</atom:title>
      <accept>image/*</accept>
    </collection>
  </workspace>
  <workspace>
    <atom:title>Side Bar Blog</atom:title>
    <collection href="http://example.org/reilly/list">
      <atom:title>Remaindered Links</atom:title>
      <accept>entry</accept>
      <categories fixed="yes">
        <atom:category scheme="http://example.org/extra-cats/" term="joke"/>
        <atom:category scheme="http://example.org/extra-cats/" term="serious"/>
      </categories>
    </collection>
  </workspace>
</service>
```

Category Documents

(39)

- Categories are important for creating and reading entries
 - they may contain metadata using any classification scheme
- [Service Documents](#) [Service Documents (1)] contain a list of allowed categories
- AtomPub defines a document format for standalone category documents
 - a useful interface between AtomPub systems and other systems using classification schemes
 - REST makes it important to make relevant resources accessible

Category Document Example

(40)

```
<app:categories xmlns:app="http://purl.org/atom/app#" xmlns="http://www.w3.org/2005/Atom"
fixed="yes" scheme="http://example.com/cats/big3">
  <category term="animal"/>
  <category term="vegetable"/>
  <category term="mineral"/>
</app:categories>
```

Extending Atom/AtomPub

Atom/AtomPub as Foundation

(42)

- Atom/AtomPub can be used as a foundation
 - apply the methodology of [RESTful Web Service Design](#) [RESTful Web Service Design (1)]
 - identify missing representations/links/states
 - add these to the Atom/AtomPub model
- Atom has a well-defined extension model (sort of)
 - *foreign markup* must be silently ignored by Atom processors
 - *foreign markup* is not allowed to change the semantics of Atom markup
 - every Atom extension must be well-behaving with plain Atom processors
- Atom can serve as a good example for extensible markup
 - if you're not using Atom, consider using [Atom's extension model](#) [http://tools.ietf.org/html/rfc4287#section-6]

39 of 45

7/4/2010 19:51

RESTful Web Service Design

(43)

- URIs are used for [resource identification](#) [What is REST?: Resource Identification (1)]
- HTTP's methods are used as the [uniform interface](#) [What is REST?: Uniform Interface (1)]
- [Representations](#) [What is REST?: Self-Describing Messages (1)] can use any syntax and structure
- [Hypermedia](#) [What is REST?: Hypermedia Driving Application State (1)] is based on links in representations
- Use transaction resources to create [stateless interactions](#) [What is REST?: Stateless Interactions (1)]

40 of 45

7/4/2010 19:51

RESTful Web Service Design Procedure

(44)

1. Figure out the data set
 2. Split the data set into resources
 3. [Name the resources with URIs](#) [What is REST?: Resource Identification (1)]
 4. Expose a subset of the [uniform interface](#) [What is REST?: Uniform Interface (1)]
 5. [Design the representation\(s\)](#) [What is REST?: Self-Describing Messages (1)] accepted from the client
 6. [Design the representation\(s\)](#) [What is REST?: Self-Describing Messages (1)] served to the client
 7. [Design hypermedia integration](#) [What is REST?: Hypermedia Driving Application State (1)] with other resources
 8. Figure out [the normal control flow](#) [What is REST?: Stateless Interactions (1)]
 9. Figure out error conditions
- (This is the ROA methodology)

Watch Caching

(45)

- Decent HTTP libraries have built-in caching
 - caching can also be implemented by proxies or reverse proxies
- HTTP has various ways to control caching
 - Expires set an expiration data on a response
 - Cache-Control for various cache control directives
 - ETag for uniquely identifying a resource version
- Caching allows *Conditional GETs*
 - If-Modified-Since is conditional based on the Last-Modified date
 - If-None-Match is conditional based on the ETag label

REST Tools

Toolboxes and Frameworks

(47)

- Toolboxes contain a number of low-level tools
 - tools for essential functionality (HTTP, URI, XML)
 - add-ons for frequent tasks (HTML tidying, XML validation, language tag parsing)
 - diagnostic and test tools (HTTP monitoring, protocol validators)
- Frameworks provide programming/structuring support
 - frameworks encourage and support a certain coding style and structure
 - REST focuses on exposing uniform interfaces and various media types
 - code should be structured by representation, not by function

REST Frameworks

(48)

- JAX-RS
- RESTlet (covers any REST, not just RESTful HTTP)
- Ruby on Rails
- Django
- GData

Poster

(49)

